Developing an Online Algorithms Tutorial for New Zealand High Schools

November 11, 2013

Caitlin Duncan
ced48@uclive.ac.nz
Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

Supervisor: Professor Tim Bell
Tim.Bell@canterbury.ac.nz

Abstract

Computer Science was introduced to New Zealand schools as an NCEA topic for the first time in 2011. The rapid development and introduction of the NCEA standards resulted in a lack of teaching resources and left many teachers feeling unprepared to teach these topics. Providing easily obtainable resources is crucial to the success of this curriculum. The research underpinning the development of a resource for students and teachers, to assist them with the algorithms section of the curriculum, is presented. "Algorithms" will form a chapter in the online textbook, the Computer Science Field Guide (CSFG). The chapter will convey the basic concepts of what an algorithm is, their associated costs and the differences in costs for algorithms which accomplish the same tasks.

An introductory video, a sorting algorithms visualisation, and two interactive applications have been developed for the chapter. These interactives use Searching and Sorting Algorithms to illustrate to students the differences in costs between different algorithms. Algorithm simulations and a preliminary teacher survey indicate the chapter content is successful and its effectiveness will be fully assessed after its release at the 2013 Computer Science for High Schools event on December 4th.

Acknowledgments

I would like to thank my excellent supervisor, Tim Bell, for his constant support, feedback and advice during the project. I would also like to thank Rhem Munro for his great work implementing the visualisation and interactives for the chapter, Jack Morgan for sharing his knowledge on the Field Guide, Michael Bell for directing and editing the video and Duncan Ballinger and Edward Dalley for volunteering their time to star in it. Last but not least, thank you to my parents for their constant support.

Contents

1 Introduction

1.1 Background

In New Zealand we are facing a shortage of trained professionals in the Computing industry. There is a high demand for Computer Science (CS) and Software Engineering graduates and many high earning jobs are going unfilled [10, 9]. This is a loss for our economy and puts significant strain on the industry. The number of graduates required in this industry has been increasing steadily for much of the 21st century. In contrast the number of students entering CS degrees has been decreasing since the year 2000. In 2005 Brislen [8] estimated that by 2010 the number of ICT professionals required in New Zealand would increase by 44,000, while the number of graduates entering the industry between 2005 and 2008 would be only 3,500. This is an issue faced by many western countries. The estimates for the United Kingdom where similarly dire with an estimated 150,000 employees needed each year and only 20,000 graduating each year [8].

There are many factors which contribute to students not choosing to enter computer related courses at university. The most significant of these is a lack of previous exposure to the subject. This results in a general misconception about the field of CS and what it involves, students lacking confidence in their own abilities in the field and a general lack of interest in this subject area. It has been observed by Margolis and Fisher [14] that an incorrect perception of the field of CS, and unpleasant experiences with general computer use, rather than the field of computing itself, influence students to reject CS as a career path. Being exposed to the field of CS during high school in a positive manner, and with study that truly reflects what the subject area is about, is an excellent way of encouraging students into further study. A lack of a suitable CS curriculum on the other hand will deter potentially interested students from the field.

In 2008 two separate reports were published which examined and evaluated the current state of computing in the New Zealand curriculum. Both reports found multiple problems with the CS curriculum. At that time the subject of computing in New Zealand schools was either nonexistent, or focused on using computer applications and file system management. It was extremely uncommon for any school classes to teach programming or to engage students in the concepts behind software design. This gave an impression to students that CS was an extension of these 'computing as a tool' ICT classes [4]. At this stage the majority of assessments available in this subject where Unit Standards which required students to do little more than use an application. These were pass/fail assessments, a style of assessment that is generally unappealing to the more academic students as there are no high grades to be achieved. The alternative to these computing unit standards was to apply Achievement Standards from the Technology curriculum to computing. Achievement Standards allow students to achieve a wider range of grades and so appeal more academically. However these standards were not devel-

oped specifically for computing and were found to encompass an inappropriately high workload for students when applied to this subject[13]. From these findings it was recommended by the authors of these two papers that CS be realised as a subject in its own right and that new Achievement Standards were needed for this subject [13, 9].

In an attempt to rectify these concerns and address these recommendations, the Ministry of Education formed the Digital Technologies Experts Panel (DTEP) with the purpose of creating a plan for improving the curriculum[4]. The panel included representatives from industry, tertiary institutes and high schools. The panel released its recommendations in May 2009 which advised the creation of a new subject area of "Digital Technologies". This subject was to include five sections related to digital technology, one of which would be Programming and Computer Science and would cover topics from both CS and Software Engineering. The DTEP also advised a very short timeline for the implementation of these standards and, as recommended, the level 1 (year 11) Digital Technologies Achievement Standards were created and made available for schools to offer in 2011. Level 2 and 3 followed in 2012 and 2013[4]. All of the Digital Technologies standards can be accessed from `http://www.nzqa.govt.nz/ncea/assessment/search.do?query=Digital+Technologies&view=all&level=01`

In New Zealand the most common high school qualification is the National Certificate of Educational Achievement (NCEA). This is offered in the final three years of high school, and year 11 corresponds to NCEA level 1, year 12 to level 2, and year 13 to level 3. NCEA is assessed through Unit Standards, which are pass/fail assessments, and Achievement Standards where students may fail, the grade being N for Not Achieved, or they can achieve one of 3 passing grades: Achieved, Merit and Excellence. The new Programming and Computer Science standards include three Achievement Standards at each of NCEA levels 1-3, which task students with designing a program, implementing a program in code, and with demonstrating their knowledge of key concepts in CS and Software Engineering. This means the new CS content covers much more than simply programming. Students who take all of these standards through their final years of school will be exposed to a range of topics including algorithms, computer graphics, human and computer interaction (HCI), machine learning and complexity. These are academically challenging subjects when studied in depth, however through the new standards students will simply be given an overview of these fields of study and will not examine these subjects in detail. The goal of the new curriculum is to expose students to these subjects so they have an idea of what the field of CS is about, it is not for them to gain an in depth understanding of any of these fields. This project focuses solely on AS91074 "Demonstrate understanding of concepts from computer science", which is a level 1 standard.

Adoption of these new standards was not compulsory for schools but a significant number did so in 2011, with 2701 students enrolling in the program implementation standard (1.46) and 1429 enrolling in the CS concepts standard (1.44). These numbers increased in 2012, perhaps reflecting an increase of interest in the subject area [6].

Due to the rapid implementation timeline for these standards there was little time for the development of adequate resources to assist with teaching the new curriculum. This left many teachers who adopted the standards with little assistance in teaching the new subject and with no knowledge of what level of work was expected from their students. Some work has been done to provide professional development to teachers. However it is difficult to make this available for all teachers nationwide, particularly those residing in rural areas far from the Universities who have been key in providing this professional development.

This, along with a general lack of experience in the computing field left the majority of teachers with an understandable lack of confidence in teaching this subject. A 2012 survey [23] of Digital Technologies teachers found that only 64% reported feeling any confidence in teaching the programming standards and only 44% felt any confidence in teaching CS. However many of the Digital Technologies teachers surveyed reported having over 10 years experience in teaching. These teachers represent an extremely valuable resource as they have the pedagogical knowledge required to successfully present this curriculum, but they need to be provided with adequate resources and professional development to help them feel confident in their ability to teach the subject content.

To help address this lack of teaching resources the Computer Science Field Guide (CSFG) [2] was created as a project within the Computer Science and Software Engineering Department at the University of Canterbury. This is an online textbook aimed at assisting with teaching the NCEA CS Achievement Standards. It is split into a range of chapters and is still being developed, but a version with a third of the chapters written (which are aimed at supporting the Level 3 Achievement Standard) has been released and used in schools for a year. It provides a student version which is publicly accessible and a teachers version, for which access must be requested, which contains extra, higher level information to assist teachers with using the guide as a classroom resource and to help them guide their students study.

1.2 Project Goal

The goal of this project was to produce the content for the Algorithms chapter of the CSFG, thus providing a teaching resource for the algorithms section of AS91074, "Demonstrate understanding of concepts from computer science". This standard has particular significance, for the majority of students, because it will be their first introduction to the concepts of CS and could have a significant impact on their conception of the field and thus their decision of whether or not to continue studying CS. AS91074 is assessed by having students submit a report about three key concepts of CS: algorithms, programming languages and usability. Algorithms has been chosen as the focus of this tutorial. Students do not have to come away from this standard with an in-depth understanding and knowledge of particular algorithms. The standard is instead focused on "the difference between algorithms, programs, and informational instructions; the kinds of steps that are used to make an algorithm; and comparing the 'cost' of different algorithms for the same problem"[5].

1.3 Report Outline

In Chapter 2 we examine algorithms in the level 1 Achievement Standard, what should be taught in the chapter, how it will be taught and other resources for teaching algorithms. In Chapter 3 the structure of the Algorithms Chapter for the CSFG is described and the resources developed for this are discussed in detail. Chapter 4 details the expected success of the Algorithms Chapter and how it will be evaluated after its release. Conclusions are presented in Chapter 5.

2 Algorithms in the new Computer Science Level 1 Achievement Standard

Before development of the chapter content began the key lessons of the chapter, and the methods with which these would be presented, were thoroughly researched. In this section we review Achievement Standard 91074, student work related to this standard, several of the key pedagogical factors which will influence the chapter content and investigate currently available resources for teaching algorithms.

2.1 The Standard

In this section we examine the aims and learning outcomes of Achievement Standard 91074, and the results of an analysis of students submitted reports for this standard. From these findings the key concepts for the tutorial are identified and several decisions regarding the tutorial content are made.

2.1.1 AS91074

Achievement Standard 91074 is a NCEA level 1 (year 11) assessment that aims to gauge students knowledge of three key areas in Computer Science: Algorithms, Programming Languages and the Usability of User Interfaces. To demonstrate their knowledge students submit a report on these three subjects. Students are then awarded one of four grades: NA for Not Achieved, A for Achieved, M for Merit or E for Excellence, the highest attainable NCEA grade. Details of the standard are publicly accessible at [18]. As this project was focused solely on the algorithms section of the standard the programming languages and usability sections will not be further discussed in this report.

The standard gives the following criteria, which students must meet to achieve these grades. For a student to achieve an Achieved grade they must meet all the Achieved criteria, for a Merit grade they must meet all Merit and Achieved criteria, and for an Excellence they must meet all Excellence, Merit and Achieved criteria.

To obtain an Achieved Grade:

1. Describe the key characteristics and roles of algorithms, programs and informal instructions
2. Describe an algorithm for a task, showing understanding of the kinds of steps that can be in an algorithm, and determine the cost of an algorithm for a problem of a particular size

To obtain Merit:

1. Explain how algorithms are distinct from related concepts such as programs and informal instructions
2. Show understanding of the way steps in an algorithm for a task can be combined in sequential, conditional, and iterative structures and determine the cost of an iterative algorithm for a problem of size n

To obtain Excellence:

1. Compare and contrast the concepts of algorithms, programs, and informal instructions
2. Determine and compare the costs of two different iterative algorithms for the same problem of size n

Each of the points for obtaining an achieved grade is essentially repeated for Merit and Excellence, but with a higher standard of work required for each of these levels, rather than additional content [5]. At an Achieved level an answer to point 1 would simply give a description of an algorithm, a program and informal instructions. At Merit level a student would have to additionally explain what makes an algorithm different from a program or a set of an informal instructions, and at Excellence level a student would have to further discuss these differences by comparing and contrasting the three concepts. To obtain an Achieved grade for point 2 a student must describe an algorithm and show understanding of the steps involved (generally by demonstrating that they have performed the algorithm) and they must report the "cost" of the algorithm for a problem of a particular size. They can do this by simply reporting the number of comparisons made, or the running time of a program implementing the algorithm, for one run of the algorithm. To obtain a Merit grade they must report the cost of the algorithm for a range of experiments, run with different values. If the student performs a range of experiments for two different algorithms (which solve the same problem) and compares and contrasts their costs then this would achieve an Excellence grade.

The key concepts from AS91074 were identified and used as the basis of the chapter content. These were as follows

1. What an algorithm is and how it differs from the related concepts of programs and informal instructions.

2. The concept that an algorithm has an associated cost, that this cost may be non-linear and is related to both running-time and asymptotic complexity.

3. That two algorithms may have different costs even if they solve the same problem and that this difference in costs can be non-linear.

A very important fact about these key concepts is that nowhere do they mention any particular algorithms that students are required to learn. It is not the aim of this assessment that students learn exactly how several different algorithms work, instead students are encouraged to learn overall concepts about algorithms.

It is also mentioned in the Assessment Specifications [19] that the descriptions in the report are produced in relation to experiences the student has had in CS. For example, for the algorithms section of the report, this would require the student to describe their own experience using an algorithm to perform a task, such as sorting a pile of books into alphabetical order. Reports which fail to reference a students own experience with an algorithm are unlikely to achieve a passing grade. This means that along with the key learning outcomes of the chapter it is important to offer opportunities for students to perform algorithms themselves, rather than simply walking them through examples.

2.1.2 Assessing High School Students

In 2012 an in depth analysis of reports submitted for AS91074 in 2011 was performed [5]. This report identified several key areas in the algorithms section that had a great impact on the grades achieved by students. The majority of students chose either sorting or searching algorithms to investigate (63% sorting and 19% searching) and many of these students earned an Achieved grade or higher for the algorithms section of the report. It was found that students who chose other algorithms, or used their own programs, were much less likely to pass the algorithms section of the report. The majority of students unknowingly limited their ability to discuss the cost difference between algorithms as they choose to only compare the costs of their algorithms for relatively small input sizes, for example $n = 10, 20, 30$. As the non-linear difference in costs for some of these algorithms only emerges when larger numbers are used, such as $n = 100$ or $n = 1000$, some students were unable to observe this relationship. About 10% of students were also unable to observe this trend as they choose to compare algorithms with the same complexity, such as Selection and Insertion sort, and so only observed a constant difference in their costs. From this it was clear that the tutorial would need to assist students in selecting what algorithms to compare.

A significant number of students seemed to struggle with the concept of the 'cost' of an algorithm. The most appropriate interpretations of an algorithms cost would, for this standard, be either the number of comparisons made by the algorithm or the length of time it took for a program implementing the algorithm to run. While many students seemed to grasp this concept there were several cases of students interpreting the cost of an algorithm as the length, in lines of code, of a program implementing the algorithm. This suggested students required some guidance in how to measure the cost of an algorithm.

Bubble sort and Quicksort were the most popular sorting algorithms used. While these have drastically

different running times, which gives students the opportunity to talk about the non-linear difference in their costs, we want to discourage the use of Bubble sort as much as possible. Bubble sort is well known among teachers but has been shown to have little pedagogical value and can be confusing for students [1]. Selection and Insertion sort are both suitable alternatives to Bubble sort as they provide just as strong a contrast with Quicksort and are more worthwhile for students to learn. Linear search and Binary search were the most popular searching algorithms used and these provided a suitable contrast for students to discuss. The choice of algorithms made a very clear difference to the quality of student reports. The algorithms which most often led to high grades were pairs of algorithms with significantly different complexities. The most successful pairs were Binary search vs Linear search (which provided a comparison of $O(\log n)$ vs $O(n)$) and Quicksort vs one of Bubble sort, Selection sort and Insertion sort (a comparison of $O(n \log n)$ vs $O(n^2)$).

To achieve the standard students need to be able to demonstrate that they understand the concepts related to algorithms, and the deeper their understanding is the higher their grade. In order to demonstrate their understanding students need to be able to personalise their work, rather than simply paraphrasing definitions from their teacher or the Internet. Many students were prevented from achieving higher grades because the lack of personalisation in their reports left the marker doubting whether the students truly understood the definitions they had repeated.

From these findings the following decisions regarding the tutorial content were made

- The two main sections of the chapter would be 'Searching Algorithms' and 'Sorting Algorithms'. Linear search and Binary search would be used as examples for the Searching section and Selection sort, Insertion sort and Quicksort would be used as examples for the Sorting section.
- Students are not required, or encouraged, to implement the algorithms themselves and use their own programs for measuring costs. Therefore implementations of each example algorithm used would need to be provided for students to download.
- Opportunities for students to experiment with the algorithms themselves needed to be provided so they are able to personalise their reports. These opportunities would be provided through interactive applications which would allow students to perform algorithms.
- The concept of a 'cost' as the number of comparisons an algorithm makes would be emphasised through the interactives, while the downloadable programs would measure it as both comparisons and time taken.
- Students needed to be encouraged to test the algorithms with large numbers so they are able to observe the non-linear differences between algorithm costs. This would be encouraged through the student version of the chapter and stated explicitly in the teacher version.

2.2 How to teach Algorithms

Students learn better when they are given the opportunity to construct knowledge themselves through experience, rather than simply learning from definitions or complete instructions [24]. It is thus desirable that through using the chapter students are given the opportunity to discover algorithms and

the differences in their costs, for themselves, rather than simply explicitly telling students how each algorithm works and the differences between them. By actively comparing algorithms in the chapter students will be encouraged to do the same in their assignments and it will assist their understanding. The interactives in the chapter provide an ideal opportunity to allow students to learn constructively. In designing these interactives it was crucial that an emphasis was placed on ensuring each of these was likely to appeal to both female and male students.

Another key tool in supporting students construction of mental models of these concepts is the use of analogies and metaphors for the use of algorithms[11]. These will be used throughout the chapter but especially during the introduction section to ensure students have begun building a mental model about algorithms before they encounter the interactives.

It has been shown that learning about two algorithms in parallel and comparing them, rather than learning them separately, contributes to students gaining a greater understanding of both the algorithms and the differences between them [21]. Through each of the Searching and Sorting sections it will be important to present the algorithms and discuss their characteristics in relation to each other, rather than viewing each as a separate entity.

2.3 Review of available teaching resources

In 2010 a review of available resources for teaching and learning Computer Science and Programming was performed which assessed the quantity and quality of these resources for the NCEA assessments [17]. It also included a list of recommended resources. The resources recommended for the algorithms section (can be found at `http://nzacditt.org.nz/resources/programming-and-cs/level-1-computer-science-concepts-concept-of-an-algorithm`) were re-examined for this project and in this section the findings are summarised.

The list included a number of text based resources which aimed to explain what an algorithm is, teach several different algorithms (most commonly searching and sorting algorithms) and discuss the differences in the costs of these algorithms. Despite these being the same topics which need to be taught for this standard these resources were commonly too complicated, in depth and long for students of this age group. Some could be suitable extensions for the top students in a class and would be beneficial for teachers to use to gain extra subject knowledge so links to these may be offered in the chapter. The content from these resources were useful for developing the content of the chapter but needed to be condensed and edited to ensure they were appropriate for students.

There are several algorithm visualisation tools and interactive tutorials included in this list. Visualisations of algorithms have been popular methods for teaching different algorithms but it has been noted that many are too complex for students to understand [17]. It was found when examining the visualisations and interactives on this list that some covered more algorithms than were necessary and used advanced language that made them unsuitable for use by school aged students. Several contained valuable information and taught the algorithms well but unfortunately were aesthetically unappealing or repetitive, which didn't engage students.

Three of the more intriguing resources are described:

- Animation of Bubble sorting CDs [20]: This is an engaging animation which shows the process of sorting a list of CDs into order using the Bubble sort algorithm. As it teaches Bubble sort this particular interactive won't be suggested for use in the chapter, however some of the techniques it uses to teach the algorithm may be of use. It places emphasis on explaining that an algorithm needs to be unambiguous and shows the steps of the algorithm as they are performed in the animation, which underlines the concept of following the instructions of the algorithm exactly.

- Sorting Algorithm Animation System (SAAS),[16]: This resource does not teach algorithms or concepts related to them. It is simply a visualisation tool which allows the user to compare the speeds, comparisons and swaps made by Bubble sort, Insertion sort, Quicksort and Selection sort. It also allows the user to change the number of items being sorted, how the items are represented and the starting state of the items (i.e. pre-sorted, random, partially sorted), all of which are characteristics which contribute to the success of algorithm visualisations [17]. This is an excellent resource for illustrating the differences in costs between algorithms, however it would not be a suitable introduction to the topic for students as it does not allow them to discover the algorithms, and the differences between them for themselves. It would be more useful as a tool for reinforcing lessons from the subject and so will be mentioned in the chapter as a resource for students to use after they have been through the chapter content.

- Sorting Algorithm Animations [15]: This webpage contains pseudocode descriptions, discussions and visualisations of a range of sorting algorithms. It allows users to compare all these algorithms, on a range of different types of lists (random, nearly sorted, few unique and reversed) and with a small range of different list sizes. Like SAAS this is an excellent resource for enforcing the lesson of different algorithms having different costs. As it does not analyse the differences between these algorithms however it could be a suitable resource to be suggested for use earlier in the chapter, rather than as an extension activity at the end.

- Brick Sorting Interactive [12]: This interactive aims to teach a range of sorting algorithms including Selection, Insertion and Quicksort, by allowing the user to compare the weights of bricks and drawing arrows between them to indicate which is the heaviest. This interactive taught the algorithms in a constructivist manner and when we observed students using the interactive they found it highly engaging. Some students described it as addictive and when we investigated their knowledge of the algorithms afterwards many of them seemed to have a very solid understanding of the algorithms. Many however also remarked that they did not like the aesthetic design of the game and pointed out several parts of the interactive they found confusing. This interactive will be recommended in the chapter and the sorting interactive we develop will be based on this.

This list also includes kinaesthetic activities from CS Unplugged [3]. The instructions for these are free to download, the activities are aimed specifically at introducing school-aged children to CS concepts and have been shown to be effective, which make them ideal resources for this standard. The CS Unplugged activities, available at `http://csunplugged.com/`, focused on teaching Searching and Sorting were examined in detail and the interactives in the chapter are based on these.

The Battleships Searching game [3] teaches Linear search, Binary search and introduces the concept of hashing using three different versions of a two player game where each player tries to find the others battleship. The details of the game can be found at `http://csunplugged.org/sites/default/`

files/activity_pdfs_full/unplugged-06-searching_algorithms.pdf. The game is played with two corresponding print outs, as shown in Figure 2.1. Each player selects one of the numbered ships to be their ship and tells the other player the number on that ship. They then take it in turns to guess, using the letters corresponding to the ships, which ship is the opponents.

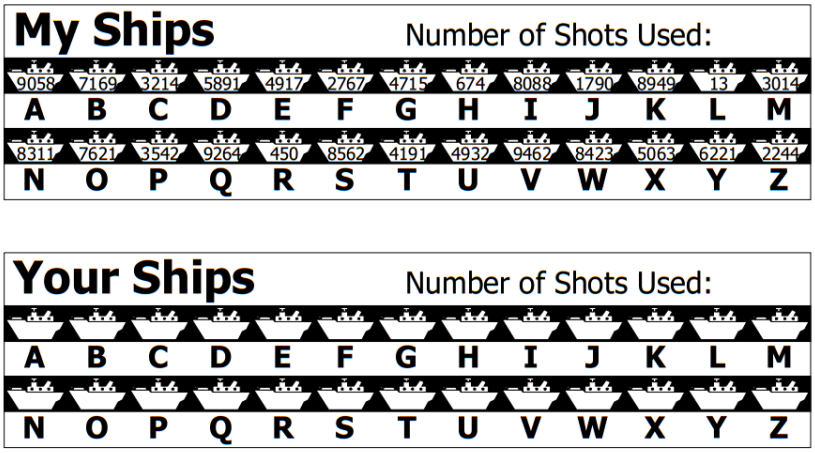


Figure 2.1: Handout for the CS Unplugged Battleships Activity

In the Linear search version of the game the numbers on the ships are in a random order and so finding the opponents ship is based solely on luck and there is no strategic way of finding the ship. This is equivalent to the cost of a Linear search. The Binary search version lists the numbers on the ship in order which allows students to employ a strategy in finding the opponent's ship. Most students will intuitively perform a Binary search by first selecting the middle ship and then, when their opponent tells them the number on that ship, they will be able to use this information to find out which half of the remaining ships the battleship they are searching for is. The hashing version groups ships based on the number on the ship which allows students to narrow down their search space before making their first guess.

The Lightest and Heaviest Sorting Weights activity can be used to teach students a range of sorting algorithms including Selection, Insertion and Quicksort. It uses a set of scales and a set of weights (which need to be visually identical), like those shown in Figure 2.2, and uses a series of instructions to prompt a student into implementing a sorting algorithm. The algorithm is then explained to the student after they have experienced using it. There are also extension opportunities included in these exercises.

2.4 The Computer Science Club

Throughout this project the author has taken part in running the Computer Science Club (CSC) at the University of Canterbury, being head tutor for the Girls Club. This is an after school club for years 7-10 students (ages 10-15) with an interest in CS and Programming. Students attending the club are given the opportunity to learn programming and many of the basic concepts of CS, including Algorithms. The head tutor role involved the organisation and presentation of learning material to the students and was an ideal opportunity to obtain valuable experience working with school-aged chil-

Figure 2.2: The scales CS Unplugged activity

dren and teaching them CS concepts. Through experience with the club the success of several learning resources, including the CS Unplugged activities and the Brick Sorting interactive, has been observed. These resources truly engaged students and contributed well to their accumulated knowledge. It has also been observed that resources providing little interaction are much less engaging for students.

3 Design and Implementation

The chapter will begin with a video and a "What's the big picture?" introductory section, each of which will aim to give an overview of the topic of Algorithms and the key concepts the chapter is going to present. The introduction section will include a algorithm visualisation showing four different sorting algorithms racing each other. The two main sections of the chapter are Searching Algorithms and Sorting Algorithms. Each of these will be made up of: a description of situations in which a computer would use these types of algorithms, why it is important that a developer designing software selects a good algorithm to use, an interactive which aims to illustrate two or three algorithms and demonstrate to students the differences in their run-times, an explanation about the algorithms in the interactives, prompts for students to think about trying the algorithms with larger numbers, and lastly links to other algorithm resources on the web and programs implementing the algorithms for students to download.

In this section the processes leading to the development of the content for the chapter are described. Section 3.1 outlines the development of the video and sorting algorithm visualisation. Section 3.2 contains the main aims of the Searching section, a description of the Present Searching interactive and the analysis for the structuring of this game. Lastly Section 3.3 describes the aims of the Sorting section and the design of the Sorting Scales interactive activity.

3.1 Video and Introduction

3.1.1 Aim of the Video

Each chapter in the CSFG begins with an introductory video. These videos aim to grab student's interest and give them an overview of the key concepts in the chapter. They are not intended to teach any of the chapter content, but by giving a 'big picture' view of the main topic they give context to the lessons in the chapter.

The first step in the video development process was to decide which concepts were to be conveyed. Several different combinations of concepts were reviewed, including the best and worst cases for an algorithm and the differences between an algorithm and a program, but the final key concepts decided on were the following.

- That an algorithm is a set of instructions for completing a task or solving a problem, we use them in our everyday lives, and algorithms are used to tell computers how to solve problems.

- There can be many different algorithms for solving a particular problem and some of these algorithms are better than others.
- Using a better algorithm can be better than using a faster computer.

3.1.2 Development

To convey these concepts it was decided that the video would involve two different algorithms being performed and their efficiency contrasted. This would allow us to introduce the definition of an algorithm and give examples of their use while also illustrating the differences that can exist in the costs of algorithms which perform the same task. Some of the situations contrasting two algorithms discussed were as follows.

- Sorting a collection of items using Selection or Insertion sort vs Quicksort. Example items to sort could be books in a pile or in a library, CDs or DVDs on a shelf, files in a filing cabinet or playing cards.
- Searching a collection of items using Linear vs Binary search. Example collections of items to search could be books in a library, CDs or DVDs on a shelf, items in aisles in a supermarket or names in a phone book.
- Solving a maze using different strategies.
- An Internet search using a 'good' algorithm and returning a page instantly vs a search using a 'bad' algorithm and taking several days to obtain results.
- Comparing a supercomputer trying to solve a problem with a bad algorithm to an old slow computer solving the same problem quickly with a better algorithm [7].
- Different algorithms involved in Networks.
- Different algorithms for how people should be boarded on airplanes.
- Following recipes.
- Finding an optimum route on a map and following the directions for this route.

The final concept decided on for the video was to use two characters, one representing a fast computer and the other a very slow computer, and have them race each other to find a book in a library. The character representing the fast computer would be much faster at looking through the books and running through the library, but would use a Linear search algorithm to try and find the book. They would start at the beginning of the library, in the A's section, and search the entire library book by book until they found the one they were searching for. The character representing the slow computer would take a much longer time to walk through the library and would examine each book for a long time before placing it back on the shelf and moving on. This character however would use a Binary search algorithm to search through the library. This slower character would find the book long before the fast character, and after checking a much smaller number of books.

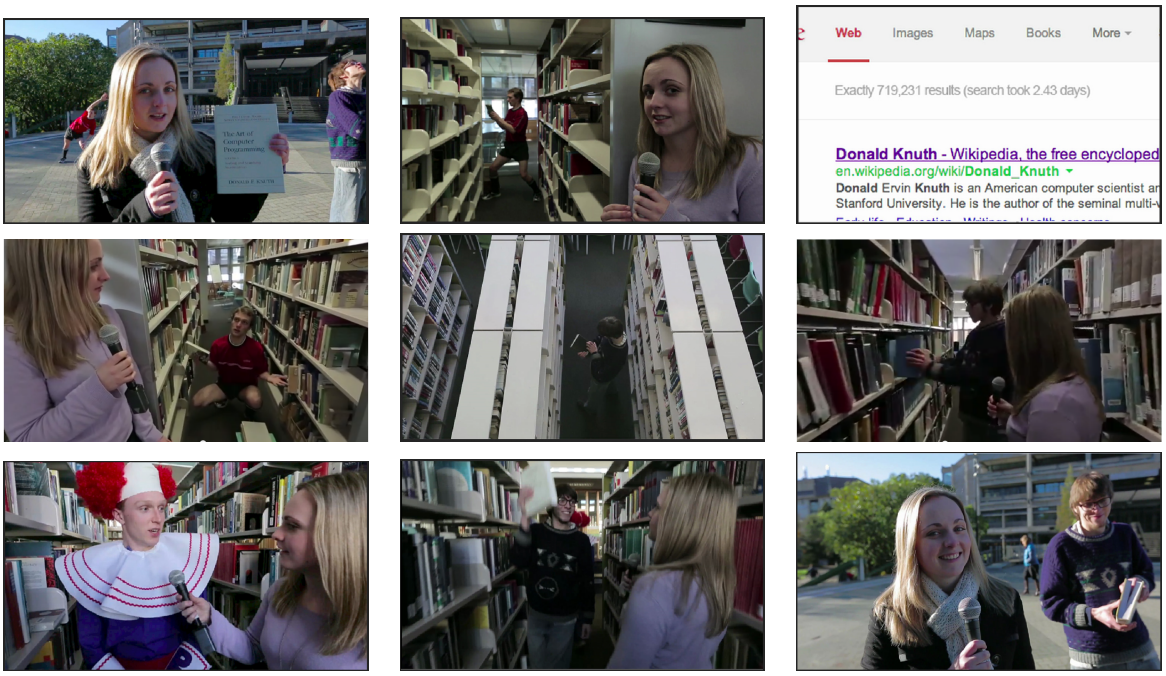


Figure 3.1: Series of images taken from the completed video

This concept was selected as it conveyed the idea that a better algorithm can be much better than a faster computer and allowed us to introduce both Linear and Binary search. The situation of searching for a book in a library is also one most students will be able to easily imagine and relate too. As the emphasis in the video is on the 'big idea' of searching it was decided that the inaccuracies between this situation and the real world would not be a problem. For example, libraries do not use an A-Z ordering for books and in practice an index would be used to locate the book. However the fictitious nature of the scenario in the video should help viewers to suspend disbelief, and after presenting the video to a group of 6 teachers and the students from the CSC there have been no comments regarding these inaccuracies.

Video plot line

The video begins outside the University of Canterbury library (images from the video are shown in Figure 3.1) with the narrator (played by the author) giving a short description of what an algorithms is, several examples of algorithms being used by computers to perform tasks and introducing the concept of different algorithms being superior to others.

NARRATOR: We use computers to do hundreds of different tasks every day, things like searching the Internet, editing an image, or finding a route on a map. But have you ever wondered how your computer figures out how to do these things? An algorithm is a step by step process that tells a computer how to do a certain task. There can

be many different algorithms for the same problem, kind of like how there can be many different recipes for a cake. But, as many of us know from experience, some recipes are a lot better than others. Lets do a little experiment and see if the same is true for algorithms

The viewer is then introduced to Speedy Spencer and Slowcoach Slade, who symbolise our new and extremely fast computer and our old and slow computer. They begin a race to locate a book in library. It is observed by the narrator that Speedy Spencer is very fast at searching through the books on the shelf and has searched many books while Slowcoach Slade has not yet checked one. However Spencer seems to be searching the books sequentially.

NARRATOR: It appears that Speedy Spencer's strategy is to employ a sequential search algorithm by starting with the first book and checking each individual book

Later in the video

NARRATOR: Alright so Spencer has almost finished his first shelf of books. I think he's got a few thousand more shelves to go before he finds the right book. It's a bit like if you took one of the fastest computers in the world and gave it one of the worst algorithms in the world. Imagine if a fast engine like Google gave you search results by searching through every page on the Internet one by one.

Video cuts to an image of Google Search with the text "Search took 2.43 days"

Slowcoach Slade on the other hand is using a Binary search to find the book.

NARRATOR: Slade seems to be employing something like a Binary search algorithm as his strategy. He has looked at a book at the centre of the library and by looking at that one book he has realised that the book he's looking for can't be in the first half of the library. So by making one 'query' he has eliminated half of the books in the library. He may be slow but he's intelligent.

Later in the video

NARRATOR: How are you going Slade?
SLADE: Pretty much as planned. I've narrowed it down and the book should be on this shelf.
NARRATOR:You already know that it's on this shelf?!
Slade checks a book in the middle of the shelf
SLADE: On this half of the shelf

By checking one book in the middle of a shelf Slade is able to eliminate half of a shelf of books. This illustrates how quickly Binary search is able to eliminate items compared with Linear search.

The video briefly introduces a new character, Bozo the Clown, who is also searching for the book using a bozo search. Bozo search, which is inspired by Bogosort, simply involves Bozo the Clown picking up books at random until he finds the right book (which could possibly never happen). This short interaction with Bozo will be used later in the chapter to introduce a 'Curiosity' aside about Bogosort and how brute force algorithms are particularly inefficient.

BOZO: Excuse me
NARRATOR: Bozo? Are you searching for the book as well?
BOZO: Yup I'm doing a Bozo search.
NARRATOR: How does that work?
BOZO: Well, I pick up a book, and hopefully it's the right book.
NARRATOR: But what if it isn't the right book?
BOZO: Then I go somewhere else and try another book. Excuse me

After the narrator speaks to Bozo the Clown it is revealed that Slowcoach Slade has found the book and has won the algorithms race. The video ends with a short conclusion outside the library which touches on the concept of the non-linear costs between the algorithms and the contents of the chapter. In this final dialogue effort was made to hint at the scalability difference between Binary search and Linear search due to the log performance of Binary search.

NARRATOR: So it looks like the algorithm you choose can make a big difference, particularly with large amounts of data. You see if the library was twice the size it would take Speedy Spencer twice as long to search it, but Slowcoach Slade would only have needed to check one extra book! In this chapter we will be looking more at what algorithms are used for, and why choosing the right algorithm makes a huge difference

The full video can be viewed online at `http://www.youtube.com/watch?v=FOwCCvHEfY0` and can be viewed or downloaded at `http://vimeo.com/69609500`. The video is provided on vimeo, as well as youtube, so that teachers are able to download it and play it in a classroom, as some schools limit access to youtube.

3.1.3 Introduction and Visualisation

After the video a short introduction section reiterates and emphasises the key lessons from the video and describes the sections of the chapter. To emphasise the points that there are a number different algorithms for the same problem and that some of these algorithms are better than others, a sorting algorithm visualisation has been designed, and was developed by Rhem Munro.

The aim of this visualisation is not to teach the algorithms, although it may be of use to refer to it again after students have learnt the sorting algorithms so they can see them in action. It is intended

to be engaging, to keep students interested in the chapter content, and to show again the difference in the performance of algorithms.

As the sorting algorithms animation website [15] already offers students the options to change the amount of data being sorted and the starting state of the data it was decided that these options would not be offered in our visualisation. A link to the sorting algorithms animation website will be provided further through the chapter after students have learnt about some of the algorithms it shows. Sending students to this website, which shows a wide range of different sorting algorithms and many different options for using these, in the introduction to the chapter may intimidate and overwhelm some students. The visualisation we have designed shows only four algorithms. We have also placed a strong emphasis on the aesthetic of the visualisation as it is intended to be eye catching and engaging. The visualisation is shown in Figure 3.2.

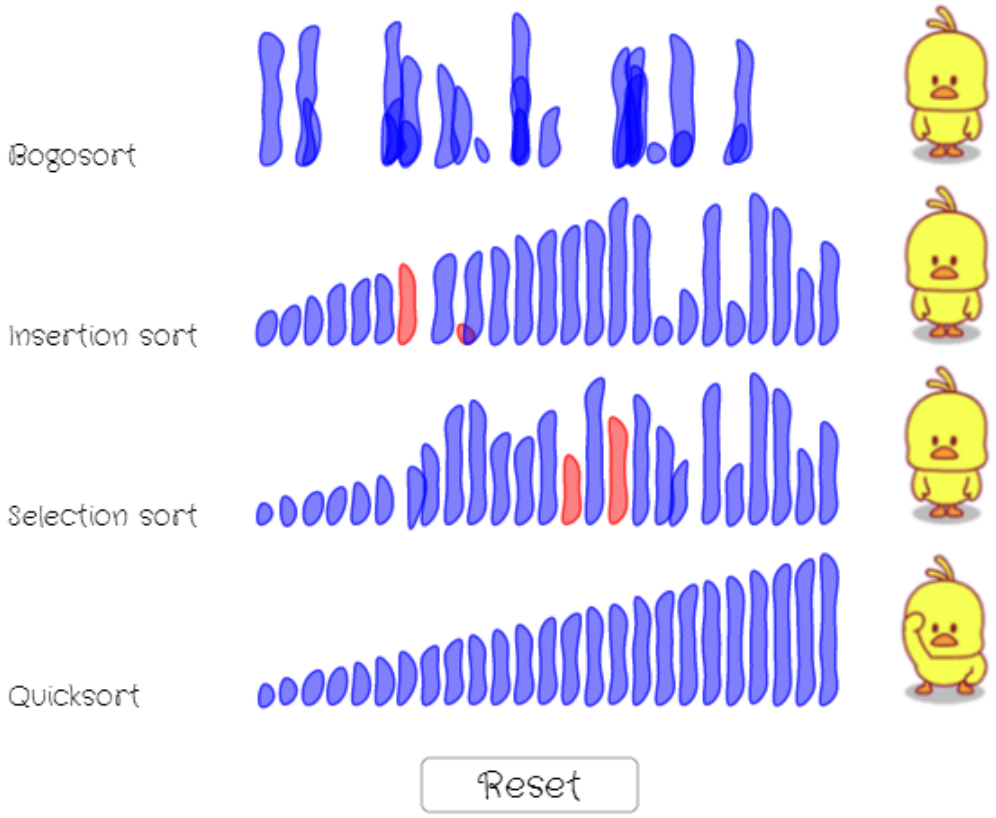


Figure 3.2: The sorting algorithms comparison animation

Each of the rows of bars are sorted into order using a different algorithm. The algorithm used for the top row is bogosort and so this row of bars is continuously shuffled until it ends up in the correct order (which is very unlikely to ever actually happen). The other rows are sorted using Insertion sort, Selection sort and Quicksort. Once the bars are found to be in the correct order the bird beside that particular row begins a victory dance, which the Quicksort bird can be seen doing in Figure 3.2. Students are able to restart the visualisation to view it again at any time.

The introduction also reiterates an important point made in the CSFG Introduction Chapter, which

states “computers can perform billions of operations every second, and yet people often complain that they are too slow. Humans can perceive delays of about one tenth of a second, and if your program takes longer than that to respond it will be regarded as sluggish, jerky or frustrating. You’ve got well under a second to delight the user!” [2]. It also explains that choosing the best algorithm is important because computers have to deal with huge amounts of information. If you don’t choose the correct algorithm to use your software will not be efficient enough and people will not use it.

3.2 Searching

3.2.1 Content/aims

The Searching algorithms section of the chapter focuses on Linear and Binary search. These algorithms were chosen because it is easy for students to gain a high level understanding of them and they are easy for students to perform themselves with physical objects. The non-linear difference in their complexities also makes them suitable choices for students to compare in their assignments and, as stated previously, students who have used these algorithms in the past have achieved well in the assessment [5].

The definitions of Linear and Binary search provided in the chapter will be no more detailed than the following:

- Linear search: You are given a list or collection of items and are asked to find a particular item. You look at the first item in the list and compare it to what you are searching for. If it is the item you are searching for then you are finished. Otherwise move on to the next item and continue until you have found the item, or you reach the end of the list
- Binary search: You are given an ordered list or collection of items and are asked to find a particular item. You look at the item in the middle of the list and compare it to the item you are looking for. If it is the item you are looking for then you are finished. If the item you pick up is ‘greater’ than the item you are looking for (according to the ordering of the list) then discard all items greater than this item. If the item you pick up is ‘less’ than the item you are looking for then discard all items less than this item. Now continue searching through the remaining half of the original list by looking at the item in the middle of the remaining list and repeating this process

More detailed descriptions of these algorithms will not be provided as students are required to describe these in their own words in their reports. Programs implementing these algorithms will be provided for students to download in Python, Scratch, JavaScript and Java. These languages are the most commonly used in high schools and so students should have little difficulty in running them [22].

3.2.2 Interactive

The Searching algorithms interactive is a game in which students have to search through a large number of presents in an attempt to find and collect the missing pets of two children. Presents that don't contain pets have monsters inside and all these monsters, as well as the pets, have a unique number on them. The student knows which number the pet has on them and so can use this knowledge in some of the levels to help them find the pet. Each time a present containing a monster is opened the player loses a life.

The game is based on the CS Unplugged "Battleships" activity, described in the related work section. It was decided that the act of blowing up battleships was likely to appeal more to male students than to female. Initially we considered developing two separate games which would function in essentially the same way, except in one the aim would be to destroy battleships and the other would have a theme targeted at a female audience. It was decided however that the division of activities by whether they were 'girl things' or 'boy things' was not something we wished to encourage and it would be much more appropriate to develop a game which was gender neutral and would hopefully appeal to all students.

3.2.3 Analysis

Linear Search levels

Levels 1 and 2 of the interactive aim to teach the concepts of Linear search. As the numbers on the monsters and pets in these levels are in a random order there is no way for students to use the information about the number on the pet to help them find it. The only strategy they can possibly use is to check a present at random, and if it does not contain the pet then choose another present, and then another and so forth. This emulates a Linear search through a list of data, even if students do not specifically check the first present, and then the next and so on.

In each of these levels students are given the same number of lives as there are presents on the screen. This is to ensure that they will always be able to find the pet. It also serves to enforce the idea that the worst case for these levels, and the Linear search algorithm, is the case where the pet they are searching for happens to be in the last present they check. In the first level students are presented with only 10 boxes to search through and so will on average have to search through 5 of the presents before they find the pet. The second level has 20 presents for students to search through and since the number of objects to search has doubled it will take students on average twice as long to find the pet as previously.

However these are simply the average cases. One student may find the pet in one guess on each level, while another may have to search through every present on one level and find the pet much faster on the next. Therefore the interactive will be backed up by explanations within the chapter of the average, best and worst cases of the algorithm. These explanations will also prompt the students to consider the situations where the number of items being searched was much larger, for example if there were 1000 presents, or maybe 1,000,000 presents.

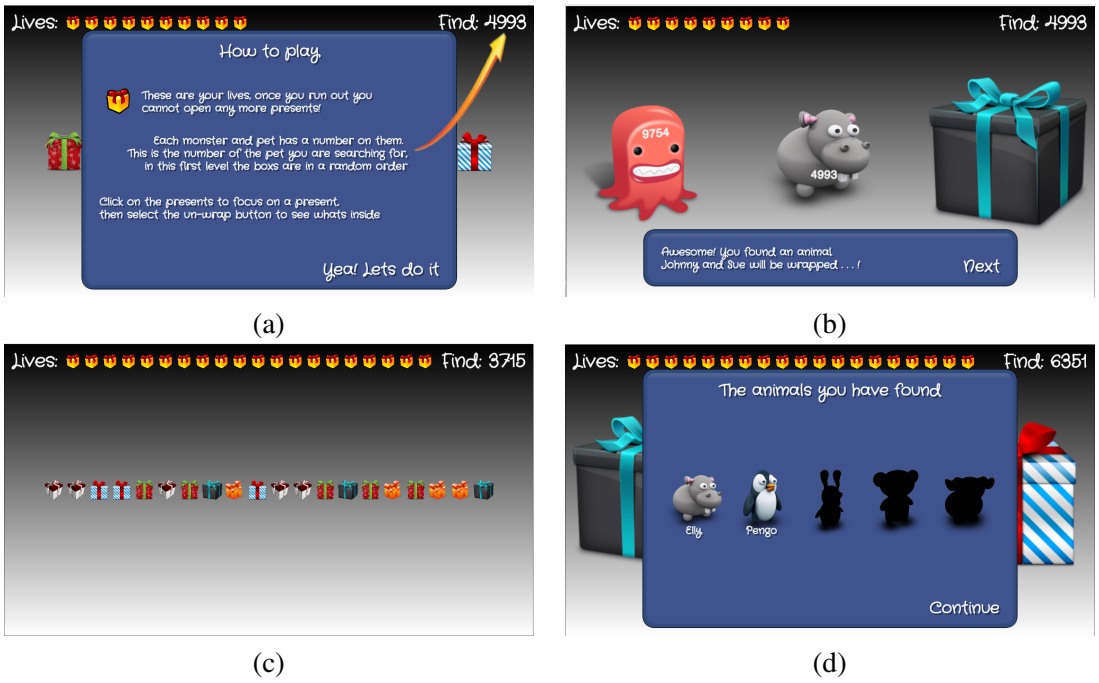


Figure 3.3: Linear search Levels. (a): When the player first enters the game they are presented with an instruction screen. (b): The player has found the first pet. (c) The second Linear search level has a larger number of boxes to search. (d) Once the player has finished the Linear search levels they will have located two pets, but there are still 3 more to find.

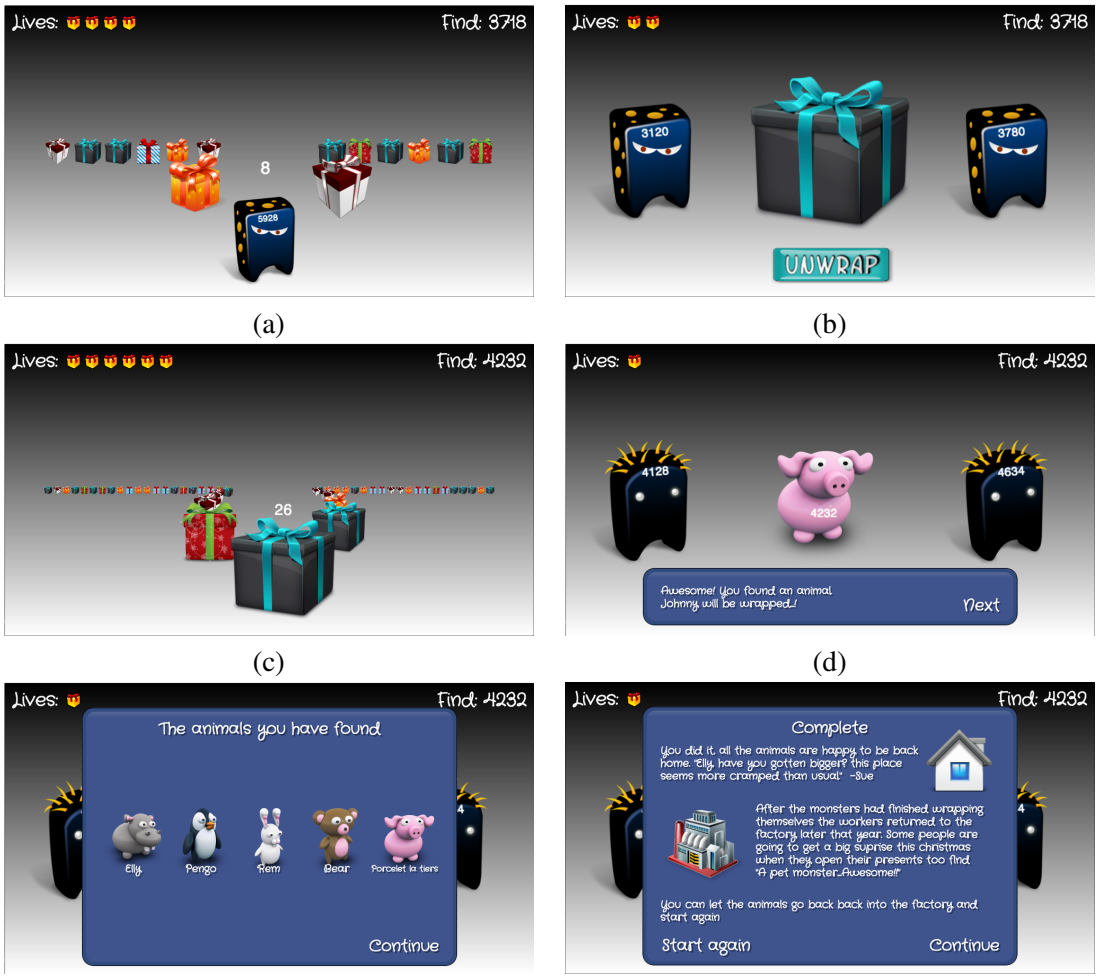
The teachers version will also include suggestions on how to run this as a class activity. If a whole class of students plays these two levels, then report to their teacher how many presents they had to open, the teacher will be able to show the class that if they average these numbers for each level that the second level will take about twice as many guesses as the first.

Binary search levels

Levels 3, 4 and 5 of the game aim to teach the concepts of Binary search and differ from the Linear search levels in two key ways. In these levels the numbers on the monsters and pets inside the presents are ordered from smallest to largest and the player is given fewer lives. They are informed of these changes before they begin the third level. As the presents are now in sorted order students are able to use the number of the pet to help them find the pet faster, and since they do not have enough lives to check every box if they do not use a Binary search they are likely to run out of lives before the pet is found.

Through the use of the CS Unplugged Battleships game and Ping Pong Ball activity [3] it has been observed that when people are asked to find a number in a sorted list they will often instinctively choose the central item and begin a Binary search without requiring any instruction or even knowing that they are using the algorithm. As the Present Searching game is in essence the same as the Battleships game

it is presumed that most students will demonstrate this behaviour.



(a) (b) (c) (d) (e) (f)

Figure 3.4: Binary search Levels. (a) The player begins by opening the present in the centre of the list. (b) The player knows the pet must be in this box. (c) The final Binary search level has 50 presents, many more than the previous levels. (d): The last pet has been found. (e) The player has collected all the animals. (f): Once all pets are found the player can choose to reset all the levels so they can play through again.

However as students know that the numbers used are four digit numbers, and therefore come from the range 1000-9999 they may use this knowledge to guess the general area in the list where the number they are searching for is. For example if they are asked to find the number 2031 they may choose a present much closer to the lower end of the presents, or if the number is 9744 they may choose one at the higher end of the list rather than the middle present. This method is known as Interpolation search. In its first state the game randomly sampled numbers from the range 1000-9999 for the numbers on the monsters and pets. As the aim of the interactive was to encourage students to use Binary search it was important to assess whether the use of Interpolation search would give better results than Binary search when numbers were selected in this way, and if it did then a method of selecting numbers

Table 3.1: Random list and index

	Binary	Interpolation	Difference
16	3.372	2.904	0.468
32	4.204	3.691	0.513
64	5.102	4.556	0.547

Table 3.2: Random list, index not in the centre

	Binary	Interpolation	Difference
16	3.486	2.837	0.649
32	4.281	3.634	0.646
64	5.167	4.548	0.619

which favours Binary search over Interpolation search needed to be found. The effects of selecting an element which was not in the centre of the list were also investigated. This was done because we wished to encourage students to repeat the method of halving a list so they developed a stronger concept of Binary search, rather than picking the middle present and finding the pet immediately.

To compare the performance of Binary search and Interpolation search on lists selected randomly, and thus investigate whether lists needed to be sampled in different ways to prevent students from missing the point of Binary search, a number of simulations were performed. These generated a random list, with no repeating elements, and selected a random element in that list to be searched for. A Binary search and a simplified Interpolation search where then used to locate the element and the number of comparisons each performed was recorded. This was repeated 100,000 times, each time a new list was generated, for lists of length 16, 32 and 64. the average of the number of comparisons taken for each list size and algorithm was recorded. The simplified version of Interpolation search functioned by calculating the most likely location of the element in the list, using the known range of 1000-9999 and assuming the numbers in the list were evenly distributed, and checking that position. If the element was not found at that position then the function continued by using a Binary search to search the remainder of the list. The results of this are shown in Table 3.1. The difference was found by subtracting the comparisons for Interpolation search from Binary search. This was repeated again with a randomly selected list, however this time the element being searched for was chosen so that it was not the centre item (or one of the two centre items) in the list. The results from this are shown in Table 3.2

Interpolation search outperformed Binary search when the items in the list are randomly selected from a known range by around half a comparison. When simulations were repeated for a range of lists lengths between 10 and 100 it was found that as the length of the list grew the difference between the performances of the two algorithms also grew. The difference also increased when the item chosen from the list was not the centre item. This was expected as this change would clearly decrease the performance of Binary search as it would always take more than 1 comparison to find the item being searched for. Surprisingly this also seemed to improve the performance of Interpolation search.

Two different techniques for selecting the numbers in the list were simulated to assess if these would be suitable for use in the interactive to help students avoid missing the point of the activity by performing an Interpolation search.

The first of these was incorporating a chance that each time a number was chosen to be added to the list the next number chosen would be within 5 integers of this number. For example if the number 3897 was added to the list then there would be a chance that the next number added to the list would be one of 3898, 3899, 3900, 3901 or 3902. This method was referred to as ‘clumping’. It was tested on lists of size 16, 32, and 64, with ‘clumping chances’ of 10%, 20% and 30%, and with lists with

randomly chosen indexes and indexes that were not in the centre of the list. In this situation the range of the list was still set to 1000-9999 so it was assumed the range was known to the Interpolation search method. The results of these simulations are shown in Tables 3.3 and 3.4.

Table 3.3: Random list and index, with Clumping

	Binary			Interpolation		
Chance	10%	20%	30%	10%	20%	30%
16	3.375	3.369	3.376	2.928	2.961	2.982
32	4.214	4.217	4.215	3.718	3.733	3.762
64	5.124	5.118	5.127	4.560	4.586	4.600

Table 3.4: Random list and index not in centre, with Clumping

	Binary			Interpolation		
Chance	10%	20%	30%	10%	20%	30%
16	3.504	3.498	3.499	2.859	2.895	2.930
32	4.275	4.273	4.273	3.652	3.683	3.700
64	5.163	5.162	5.162	4.543	4.544	4.563

This method however did not yield significantly different results from a random selection.

The second method, which will be referred to as Subset range, simulated took a subsection of the possible range and sampled numbers from this new range. For the simulations we choose to set the size of the new range to 400. Therefore each time a new list was created a random number between 1400 and 9999 was chosen and set as the maximum for the new range. The minimum was then set to the maximum-400, and a list of numbers was randomly sampled from this range. As the only information provided to students would be that the numbers are four digit numbers they will assume the range to be 1000-9999, and so the Interpolation search also assumed this range. The simulations were run again with lists of length 16, 32, and 64, and were tested with a random index and a non centre index. The results of these Subset range simulations are shown in Tables 3.5 and 3.6

The Subset range method proved successful with Binary search achieving better results than Interpolation search even when the element being searched for was never the centre element in the list.

This method was further tested with the list lengths 15, 25 and 50, as these were the lengths chosen to be used in the Binary search levels for the interactive. It was also tested with a different method for selecting the item being searched for. This new method, which will be referred to as Selective Index,

Table 3.5: Subset range and random index

	Binary	Interpolation
16	3.379	3.621
32	4.218	4.497
64	5.122	5.422

Table 3.6: Subset range, index not in centre

	Binary	Interpolation
16	3.503	3.599
32	4.295	4.469
64	5.178	5.412

selects an item that will not be found in the first two checks of the list if a Binary search is used. It also accounts for the situation where students chose a number that is roughly in the middle of the list. For example say the list of 15 items in Table 3.7 represent 15 presents. The present chosen to contain the number being searched for will not be any of the presents shown in bold.

Table 3.7: Numbers which will not be searched for

1	2	3	**4**	5	6	**7**	**8**	**9**	10	11	**12**	13	14	15

These simulations gave the following results shown in Table 3.8. Interpolation outperformed Binary search for lists of length 15 and 25, however binary proved better for the largest length of 50. These differences are very small and as students will be forced to repeat the process of halving a list at least twice with this approach it was decided to use Selective Index with a Subset range to select the numbers for the interactive.

Table 3.8: Selective Index with Subset range

	Binary	Interpolation	Difference
15	3.600	3.495	0.105
25	4.222	4.139	0.083
50	5.045	5.063	-0.018

3.3 Sorting

3.3.1 Content/aims

The Sorting Algorithms section of the chapter focuses on Selection sort, Insertion sort and Quicksort. Similar to the algorithms chosen for the Searching section, these algorithms were selected for their different run times (for Selection or Insertion sort vs Quicksort) and the successful results achieved by student using them. Selection and Insertion sort are very simple algorithms to explain and demonstrate with physical objects. Despite the complexity of implementing Quicksort it can also be simple to teach the basic method and demonstrate it with objects, which is all students require for this assessment.

The definitions of Selection, Insertion and Quicksort provided in the chapter will be no more detailed than the following:

- Selection sort: You are given an unordered list or collection of items and asked to sort it into order. You are going to do this by first 'selecting' the largest item in the list (according to the ordering of the objects), and then finding the next largest item from the remaining items, and then the next largest, until the list is in order. To do this first pick up an item from the list. Compare this to another item in the list and if the item you are holding is larger hold on to it, otherwise swap it for the other item. Repeat this for each item in the list until you have found the largest item and set this to one side. Now choose an item from the list and repeat the process of comparing it to each item in the remaining list until you have found the second largest. Place the second largest beside the largest. Continue until the entire list has been sorted.

- Insertion sort: You are given an unordered list or collection of items and asked to sort it into order. Remove an item from the list and place it to one side, this is going to be the ordered list. Next select an item from the remaining unsorted list and 'insert' it into its correct place in the ordered list. Do this by comparing it to the item already in the ordered list. Repeat this process of removing items from the unordered list and inserting them into the ordered list until all items have been sorted into order.
- Quicksort: You are given an unordered list or collection of items and asked to sort it into order. Pick an item from the list at random. Compare every other item in the collection to this item, place the items that are smaller in a pile on your left and the items that are larger in a separate pile on your right. Place the item you chose in the middle, this item is now in its correct place. Now choose one of the piles and repeat this process. Keep repeating this process on each of the piles you make until each pile only has one item in it. The items should now be in order.

More detailed descriptions of these algorithms will not be provided as students are required to describe these in their own words in their reports. Like the Searching algorithms, programs implementing these algorithms will be provided for students to download in Python, Scratch, JavaScript and Java.

3.3.2 Interactive

The Sorting Algorithms interactive is an activity in which students sort a range of boxes into order according to the boxes weights. The comparison interaction is based on Sorting Weights Unplugged activity. The user is presented with a set of scales on to which they can drag objects to compare their weights. The interactive does not explicitly tell students a sorting algorithm and ask them to follow it. Instead it prompts the student to discover the algorithms themselves in a similar way to the Brick Sorting game and the Sorting Weights activity.

In designing the interactive we considered several different scenarios to make the activity more engaging. For example, the user would be presented with a number of gift boxes on a table and told that they were prizes for a contest that had been mixed up, the user would then have to sort the boxes into order (assuming that the prize for first place was the heaviest box). The design needed to ensure that all the objects being sorted where visually distinct, but the differences in their appearances did not suggest anything about their weight. The majority of game scenarios discussed did not meet these requirements or did not seem completely logical, for example in real life it would be very unlikely for prizes to be ordered by weight rather than monetary value. It was also decided that the weights must be selected so that whenever two are placed on the scales the difference between their weights is great enough that there is clearly a heavier weight and a lighter weight. This binary result will hopefully prevent students from guessing the position of weights due to boxes weighing almost the same amount.

It was decided that as long as the action of moving the objects on the screen was engaging then simply using coloured boxes as the objects to be sorted would be enough. To achieve this the interactive was implemented with realistic physics so all the objects in the scene react to each other. This also allows the boxes to be stacked which is particularly useful for performing Quicksort. The basic layout of the interactive is shown in Figure 3.5. This interactive is still in development but when completed the user

Figure 3.5: The Sorting Interactive

will be able to place the boxes on ledges along the left side of the screen, as shown in Figure 3.6, and then check if they have ordered them correctly using a "Test Order" button.

Selection and Insertion sort

Each of these levels will begin with only the topmost of the ledges visible. Once a box is placed on this ledge the next ledge down will slide out from the side of the screen. This is to encourage students to work on 'building up' their sorted list so they are more likely to follow the algorithms correctly. Ledges will not disappear if boxes are removed, as this would make Insertion sort difficult to perform.

Students will be prompted to perform Selection sort by dialogues which will ask them to first identify the heaviest box and place that on the uppermost ledge. After the next ledge appears they will be prompted again to find the next heaviest box and then will be left to complete the list themselves.

To encourage students to perform Insertion sort they will be prompted to randomly compare two boxes, place the heavier on the topmost ledge and place the lighter on the next ledge, once it appears. They will then be prompted to choose another box at random and place it on the correct ledge by comparing it to the two already sorted boxes. The student will then be prompted to continue this process until the boxes are all placed on a ledge.

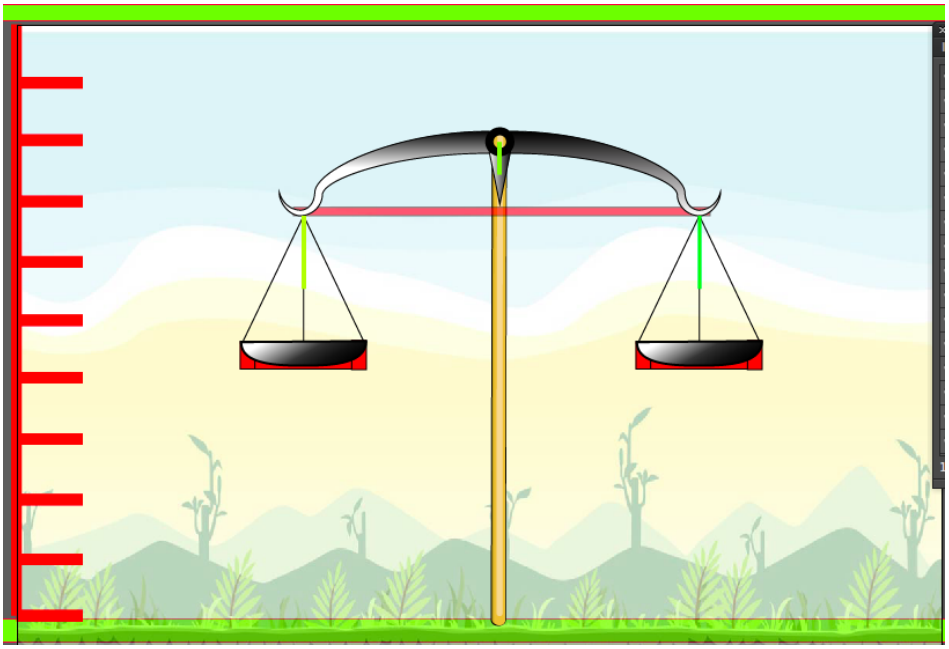


Figure 3.6: The Sorting Interactive in development

Quicksort

In the Quicksort level all ledges will be present for the entire activity so that students are able to place boxes into any position in the ordered list at any time. They will be prompted to choose a box at random and compare all other boxes to it and place these boxes into two separate piles depending on whether they are heavier or lighter than the chosen box. They should then place the box they compared all the other blocks to onto the appropriate ledge. Once a box has been placed on one of the ledges they will be told to repeat this method on one of the piles and continue using this strategy until all boxes have been sorted.

Free Sort

A version of the interactive with all ledges present and no prompts will also be made available. This is to allow students to try the algorithms again in a less structured environment where they do not have to read the prompts each time they wish to sort the boxes. It will also provide a resource for teachers to use if they wish to try the CS Unplugged Sorting Weights activity, but do not have access to a set of scales.

3.3.3 Analysis

One of the key lessons for students to learn from this activity was that Quicksort is a superior algorithm to Selection and Insertion sort. However the worst case of Quicksort will require the same number

of comparisons as Selection sort and more comparisons than the average case for Insertion sort. This meant that when students used the interactive there would be a chance that they may perceive Selection sort and Quicksort to be equal, and Insertion sort to be superior to Quicksort.

Investigating the probability distribution of Quicksort determined whether the Quicksort interactive needed to be 'rigged' to prevent students from experiencing its worst-case the first time they encounter Quicksort.

To do this two versions of Quicksort were implemented. One which performed a true Quicksort and another, which will be refereed to as 'Cheating Quicksort', which is ensured to choose its first pivot value from the centre 50% of the data. For example when given a list of the numbers 1 to 15, in a random order, Cheating Quicksort will select its first pivot from the sublist (5, 6, 7, 8, 9, 10, 11). This prevents it from encountering the worst-case for Quicksort. After the first partition has been performed Cheating Quicksort continues in the same way as normal Quicksort. Each of these algorithms was used to sort lists of different lengths and the results for 1000 trials of each list size are shown in Figure 3.7

From these results it was clear that the chances of students encountering a case of Quicksort which required a large number of comparisons was too high for list lengths of 10 or more. However list lengths shorter than these did not provide an adequate difference in comparisons between each of the sorting algorithms. Therefore it was decided to 'rig' the Quicksort level of the Sorting interactive. When a user first enters the interactive none of the boxes will have weights assigned, but there will be a list of weights from which the box weights will be chosen. Once the user places a box on the scales that box will be assigned a weight from the centre 50% of weights. All other boxes will then randomly be assigned weights from the remaining list.

This will ensure that students do not encounter the worst case of Quicksort in this interactive, while still being asked to sort a large enough number of items so that the differences between each of the sorting algorithms is clear.

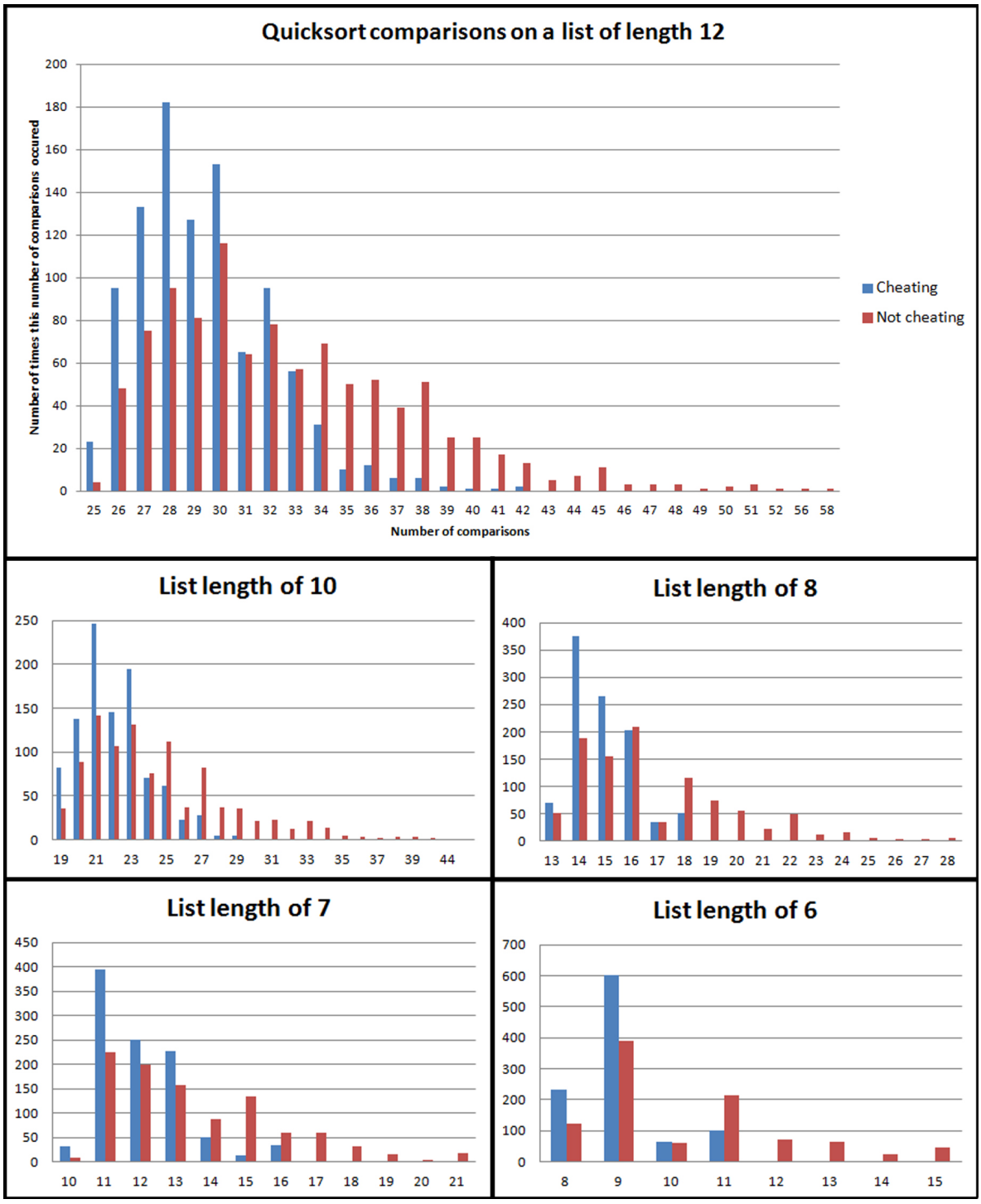


Figure 3.7: Quicksort and Cheating Quicksort Simulation Results

4 Release, Evaluation and Future Work

The complete chapter content will be released to the public at the 2013 Computer Science for High Schools (CS4HS) event on December 4th. This will be the first opportunity to gather feedback on the full chapter. At CS4HS it will be presented to a large group of New Zealand Digital Technologies teachers who will have the opportunity to review its content. This will provide valuable feedback on the likelihood of the chapters success in schools. Revisions to the chapter content can then be made if feedback indicates these are necessary.

It is however expected at this stage that the chapter will be successful. This is due to the success of the CS Unplugged activities on which the interactives in the chapter are based, experience with the Computer Science Club, the results of the simulations conducted and the decisions made from their analysis, and the results collected from a preliminary teacher survey conducted during development.

A group of six Digital Technologies teachers viewed the completed video and a demo version of the Present Searching game and volunteered to complete a survey on their reactions to this content. The demo version of the Present Searching game was essentially the same as the finished version but without the dialogues explaining the game, instead each level was explained to them by the author. Teachers where asked how much they agreed with a range of questions, with possible responses being *Strongly Agree, Agree, Neutral, Disagree* and *Strongly Disagree*. The questions and results from this survey were as follows:

1. *Did you think the video was suitable as an introduction to the topic for a year 11 class?* :
   Strongly Agree: 4 , Agree: 2 , Neutral: 0, Disagree: 0, Strongly Disagree: 0
2. *Do you think this video will function well as a 'hook' to get students interested in the subject?*:
   Strongly Agree: 2 , Agree: 4 , Neutral: 0, Disagree: 0, Strongly Disagree: 0
3. *Do you think year 11 students will find the Linear and Binary search games enjoyable?*:
   Strongly Agree: 5 , Agree: 1 , Neutral: 0, Disagree: 0, Strongly Disagree: 0
4. *Do you think these games are a good introduction to Searching algorithms, and will assist students in understanding the differences between these two algorithms?*:
   Strongly Agree: 3 , Agree: 3 , Neutral: 0, Disagree: 0, Strongly Disagree: 0

The teachers were also given the opportunity to explain their answers, to offer general comments, and to offer their opinions on how the Present Searching game could be improved. Comments received were entirely positive, excluding one which remarked that the current version of the Present Searching game did not function on an iPad. Half of the teachers remarked that the familiar environment of searching for a book in a library contributed greatly to the video's success as students would be able to relate to it. The majority of comments related to improving the Present Searching game suggested elements that could be added to make the game more exciting and since this survey a storyline has been added to the game in an attempt to address these comments. The general reaction of these teachers was that the content would function well to engage students and to introduce them to the concepts needed for AS91074.

There are several more chapters in the Computer Science Field Guide which need to be developed in the future. Although these will have different content the process which was followed to create the content for the Algorithms chapter can be used for subsequent sections. The Achievement Standards related to the content, and previous student work in these standards, will be analysed to inform the key concepts of the chapters and interactives can be designed to provide opportunities for students to learn constructively as they make their way through the chapters.

5 Conclusions

To rectify issues within the computing curriculum in New Zealand, new Computer Science and Programming NCEA Standards were created by the Ministry of Education and released in 2011. These standards are aimed at introducing high school students to the field of Computer Science and educating them about the kinds of problems Computer Scientists and Software Engineers solve. However due to the rapid implementation of these standards, and the previous absence of this subject from schools, resources for teachers and students have not been fully developed. This has left many of the teachers from schools who chose to adopt these standards feeling unprepared to teach the subject and with little confidence in their own abilities.

To assist education providers with implementing this new curriculum the University of Canterbury Computer Science and Software Engineering Department created the Computer Science Field Guide (CSFG)[2]. This online textbook will, when fully developed, provide information for students and teachers on each of the Computer Science subjects in the NCEA standards.

The aim of this project was to research and produce the content for the Algorithms chapter of the CSFG. This chapter is intended to assist students with the Algorithms section of AS91074, a Level 1 Computer Science standard. In the process of this AS91074 and previous student submissions for this standard have been analysed in order to identify the key concepts for the chapter. This resulted in the main chapter content being divided into two sections, Searching Algorithms and Sorting Algorithms, each of which use several algorithms to teach the key lessons of the chapter. In this process we have developed: an introductory video to give students an overview of the subject and grab their interest, a sorting algorithm visualisation which demonstrates several algorithms racing, a Searching Algorithms interactive game, which aims to demonstrate to students the difference in costs for Linear and Binary search, and finally a Sorting Algorithms interactive activity, which aims to demonstrate to students the difference in costs for Selection sort, Insertion sort and Quicksort.

From the analysis of several simulations and the results of a preliminary teacher survey, this content is expected to be successful in engaging students and exposing them to the key concepts related to the field of Algorithms and required for their AS91074 reports. We also believe it will provide an enjoyable and interesting introduction to the field of CS as a whole and so will potentially influence students to continue study in this area.

Further feedback on the completed chapter and the developed content will be attainable after the chapter is released to the public at the Computer Science for High Schools 2013 conference.

Bibliography

[1] Owen Astrachan. Bubble Sort: an Archaeological Algorithmic Analysis. In *ACM SIGCSE Bulletin*, volume 35, pages 1–5. ACM, 2003.

[2] Tim Bell and Jack Morgan. Computer Science Field Guide, November 2013. URL `http://www.cosc.canterbury.ac.nz/csfieldguide`.

[3] Tim Bell, Ian H. Witten, and Mike Fellows. *Computer Science Unplugged: Off-line activities and games for all ages*. Citeseer, 1998.

[4] Tim Bell, Peter Andreae, and Lynn Lambert. Computer Science in New Zealand High Schools. In *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103*, ACE '10, pages 15–22, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc. ISBN 978-1-920682-84-2. URL `http://dl.acm.org/citation.cfm?id=1862219.1862223`.

[5] Tim Bell, Heidi Newton, Peter Andreae, and Anthony Robins. The introduction of Computer Science to NZ High Schools: an analysis of student work. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, WiPSCE '12, pages 5–15, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1787-0. doi: 10.1145/2481449.2481454. URL `http://doi.acm.org/10.1145/2481449.2481454`.

[6] Tim Bell, Peter Andreae, and Anthony Robins. A case study of the introduction of computer science in nz schools. In *ACM Transactions on Computing Education*, page to appear, 2013.

[7] Jon Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, September 1984. ISSN 0001-0782. doi: 10.1145/358234.381162. URL `http://doi.acm.org/10.1145/358234.381162`.

[8] Paul Brislen. Ict has no mana: Biggs. 2005. URL `http://www.computerworld.co.nz/article/499858/_ict_has_no_mana_biggs/`.

[9] Tim Carrell, Vilna GoughJones, and Karen Fahy. The future of Computer Science and Digital Technologies in New Zealand secondary schools: Issues of 21st teaching and learning, senior courses and suitable assessments, August 2008. `http://dtg.tki.org.nz/content/download/670/3222/file/Digital%20Technologies%20discussion%20paper.pdf`.

[10] Tony Clear and Graham Bidois. Fluency in information technology-fitnz: an ict curriculum meta-framework for new zealand high schools. 2005.

[11] Michal Forišek and Monika Steinová. Metaphors and analogies for teaching algorithms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 15–20, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1098-7. doi: 10.1145/2157136.2157147. URL `http://doi.acm.org/10.1145/2157136.2157147`.

[12] David Gale, Michelle Lin Johnson, and Joseph Johnson. Sorting bricks and sticks. URL `http://mathsite.math.berkeley.edu/sorting/brick.html`.

[13] Gordon Grimsey and Margot Phillipps. Evaluation of Technology Achievement Standards for use in New Zealand Secondary School Computing Education: A critical report, April 2008. `http://www.iitp.org.nz/news/uploads/PDFs/200805NCEAReport.pdf`.

[14] Jane Margolis and Allan Fisher. *Unlocking the clubhouse: Women in computing*. The MIT Press, 2003.

[15] David R. Martin. Sorting Algorithm Animations, 2007. URL `http://www.sorting-algorithms.com/`.

[16] Neil Munday. Sorting Algorithm Animation System (saas), March 2003. URL `http://www.mundayweb.com/progs/applets/saas/`.

[17] S Murugesh, T Bell, and A McGrath. A review of computer science resources to support ncea. In *First annual conference of Computing and Information Technology Research and Education NZ (CITRENZ2010)*, pages 173–181, 2010.

[18] NZQA. Achievement standard, digital technologies 1.44: Demonstrate understanding of basic concepts from computer science, jan 2011. `http://www.nzqa.govt.nz/nqfdocs/ncea-resource/achievements/2011/as91074.pdf`.

[19] NZQA. Achievement standard, digital technologies 1.44: Assessment specification 2013, mar 2013. `http://www.nzqa.govt.nz/nqfdocs/ncea-resource/specifications/2013/level1/91074-spc-2013.pdf`.

[20] University of Washington BENEFIT (Fluency with Information Technology). Algorithms - Bubble Sort. URL `http://courses.washington.edu/benefit/FIT100/Lessons/Lesson4/graphics/cdsort.swf`.

[21] Elizabeth Patitsas, Michelle Craig, and Steve Easterbrook. Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the ninth annual international ACM conference on International computing education research*, ICER '13, pages 145–152, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2243-0. doi: 10.1145/2493394.2493409. URL `http://doi.acm.org/10.1145/2493394.2493409`.

[22] David Thompson and Tim Bell. Adoption of new Computer Science high school standards by New Zealand teachers. In *The 8th Workshop in Primary and Secondary Computing Education (WiPSCE 2013)*, page to appear, 2013.

[23] David Thompson, Tim Bell, Peter Andreae, and Anthony Robins. The Role of Teachers in Implementing Curriculum Changes. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 245–250, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445272. URL `http://doi.acm.org/10.1145/2445196.2445272`.

[24] Barry J Wadsworth. *Piaget's theory of cognitive and affective development: Foundations of constructivism* . Longman Publishing, 1996.

A Appendix

Draft Chapter

The draft Algorithms Chapter can be viewed at `http://www.cosc.canterbury.ac.nz/csfieldguide/dev/dev/Algorithms.html`. Please be aware that this is still in development and the content is being continually updated. At times it may be unavailable due to updates.